Express Mail Label No. EL640249163US 10-05-00 A

UTILITY PATENT APPLICATION TRANSMITTAL
(Small Entity)
(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.	3728-109US
Total Pages in this Submission	564

JC887 U.S. PTO 10/04/00

JC925 U.S. PTO 09/679456 10/04/00

TO THE ASSISTANT COMMISSIONER FOR PATENTS
Box Patent Application
Washington, D.C. 20231

Transmitted herewith for filing under 35 U.S.C. 111(a) and 37 C.F.R. 1.53(b) is a new utility patent application for an invention entitled:

SELF-DESCRIBING FILE SYSTEM

and invented by:

STEPHEN A. RAGO

If a CONTINUATION APPLICATION, *check appropriate box and supply the requisite information:*

☐ **Continuation** ☐ **Divisional** ☐ **Continuation-in-part (CIP)** of prior application No.: ______

Which is a:

☐ **Continuation** ☐ **Divisional** ☐ **Continuation-in-part (CIP)** of prior application No.: ______

Which is a:

☐ **Continuation** ☐ **Divisional** ☐ **Continuation-in-part (CIP)** of prior application No.: ______

Enclosed are:

Application Elements

1. ☒ Filing fee as calculated and transmitted as described below
2. ☒ Specification having 18 pages and including the following:
   - a. ☒ Descriptive Title of the Invention
   - b. ☒ Cross References to Related Applications *(if applicable)*
   - c. ☐ Statement Regarding Federally-sponsored Research/Development *(if applicable)*
   - d. ☐ Reference to Microfiche Appendix *(if applicable)*
   - e. ☒ Background of the Invention
   - f. ☒ Brief Summary of the Invention
   - g. ☒ Brief Description of the Drawings *(if drawings filed)*
   - h. ☒ Detailed Description
   - i. ☒ Claim(s) as Classified Below
   - j. ☒ Abstract of the Disclosure

UTILITY PATENT APPLICATION TRANSMITTAL **(Small Entity)** *(Only for new nonprovisional applications under 37 CFR 1.53(b))*	Docket No. **3728-109US**
	Total Pages in this Submission 564

Application Elements (Continued)

3. ☒ Drawing(s) *(when necessary as prescribed by 35 USC 113)*
   - a. ☐ Formal   b. ☒ Informal   Number of Sheets **8**
4. ☒ Oath or Declaration
   - a. ☒ Newly executed *(original or copy)*   ☐ Unexecuted
   - b. ☐ Copy from a prior application (37 CFR 1.63(d)) *(for continuation/divisional application only)*
   - c. ☒ With Power of Attorney   ☐ Without Power of Attorney
   - d. ☐ *DELETION OF INVENTOR(S)* Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. 1.63(d)(2) and 1.33(b).
5. ☐ Incorporation By Reference *(usable if Box 4b is checked)* The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.
6. ☐ Computer Program in Microfiche
7. ☐ Genetic Sequence Submission *(if applicable, all must be included)*
   - a. ☐ Paper Copy
   - b. ☐ Computer Readable Copy
   - c. ☐ Statement Verifying Identical Paper and Computer Readable Copy

Accompanying Application Parts

8. ☒ Assignment Papers *(cover sheet & documents)* **(and check for $40.00)**
9. ☐ 37 CFR 3.73(b) Statement *(when there is an assignee)*
10. ☐ English Translation Document *(if applicable)*
11. ☒ Information Disclosure Statement/PTO-1449   ☒ Copies of IDS Citations
12. ☐ Preliminary Amendment
13. ☒ Acknowledgment postcard
14. ☒ Certificate of Mailing
    - ☐ First Class   ☒ Express Mail *(Specify Label No.)*: **EL640249163US**

UTILITY PATENT APPLICATION TRANSMITTAL (Small Entity) *(Only for new nonprovisional applications under 37 CFR 1.53(b))*	Docket No. 3728-109US
	Total Pages in this Submission 564

Accompanying Application Parts (Continued)

15. ☐ Certified Copy of Priority Document(s) *(if foreign priority is claimed)*

16. ☒ Small Entity Statement(s) - Specify Number of Statements Submitted: 1

17. ☐ Additional Enclosures *(please identify below)*:

Fee Calculation and Transmittal

CLAIMS AS FILED

For	#Filed	#Allowed	#Extra	Rate	Fee
Total Claims	15	- 20 =	0	x	$0.00
Indep. Claims	2	- 3 =	0	x	$0.00
Multiple Dependent Claims (check if applicable) ☐					$0.00
				BASIC FEE	$355.00
OTHER FEE *(specify purpose)*	Assignment Recordation Fee				$40.00
				TOTAL FILING FEE	$395.00

☒ A check in the amount of $355.00 to cover the filing fee is enclosed.

☒ The Commissioner is hereby authorized to charge and credit Deposit Account No. 23-3040 as described below. A duplicate copy of this sheet is enclosed.

- ☐ Charge the amount of as filing fee.
- ☒ Credit any overpayment.
- ☒ Charge any additional filing fees required under 37 C.F.R. 1.16 and 1.17.
- ☐ Charge the issue fee set in 37 C.F.R. 1.18 at the mailing of the Notice of Allowance, pursuant to 37 C.F.R. 1.311(b).

Dated: October 4, 2000

[illegible]
Signature

Richard C. Woodbridge, Esq.
Woodbridge & Associates, P.C.
P.O. Box 592
Princeton, NJ 08542

cc: Stephen A. Rago

P01USML/REV03

JC925 U.S. PTO 09/679456 10/04/00

EXPRESS MAIL CERTIFICATE

ATTORNEY DOCKET NUMBER:
3728-109US

"Express Mail" Mailing Label Number: **EL640249163US**

"Express Mail" Corporate Account Number: **X085783**

Date of Deposit: **October 4, 2000**

TYPE OF DOCUMENTS

☐	PCT Patent Application & Request	☐	Amendment & Response to Office Action
☒	**U.S. Patent Application Specification – 18 pages**	☒	**Declaration & Power of Attorney – 3 pages**
☐	U.S. Provisional Patent Application	☐	________ Month/Day Extension of Time
☒	**Verified Statement of Small Entity Status – 2 pages**	☐	Preliminary Amendment
☒	**Assignment & Recordation Cover Sheet – 2 pages**	☐	Trademark Application
☒	**Check in the Amount of $355.00**	☒	**Check in the Amount of $40.00**
☒	**Acknowledgment Postcard**	☒	**Transmittal Letter/ Cover Sheet – 3 pages**
☐	Notice of Opposition	☒	**Drawings - 8 pages**
☒	**Information Disclosure Citation Statement – 2 pages**	☒	**Copies of IDS Citations – 7 total**
☐		☒	**Copies of IDS Other Documents – 5 total**

I hereby certify that the enclosed documents are being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 C.F.R. §1.10 on the date indicated above and is addressed to the Assistant Commissioner of Patents and Trademarks, Washington, D.C. 20231.

Typed Name:
Richard C. Woodbridge, Esq.

Signature: [illegible]

Woodbridge & Associates, P.C.
P.O. Box 592
Princeton, NJ 08542-0592
Tel (609) 924-3773
Fax (609) 924-1811

09679456 .100400

VERIFIED STATEMENT (DECLARATION) CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) AND 1.27 (c)) - SMALL BUSINESS CONCERN

Docket No. 3728--109US

Serial No.	Filing Date	Patent No.	Issue Date
	Herewith		

Applicant/ Patentee: STEPHEN A. RAGO

Invention:

SELF-DESCRIBING FILE SYSTEM

I hereby declare that I am:

☐ the owner of the small business concern identified below:

☒ an official of the small business concern empowered to act on behalf of the concern identified below:

NAME OF CONCERN: CrosStor Software, Inc.

ADDRESS OF CONCERN: 4041 Hadley Road, South Plainfield, New Jersey 07080

I hereby declare that the above-identified small business concern qualifies as a small business concern as defined in 13 CFR 121.3-18, and reproduced in 37 CFR 1.9(d), for purposes of paying reduced fees under Section 41(a) and (b) of Title 35, United States Code, in that the number of employees of the concern, including those of its affiliates, does not exceed 500 persons. For purposes of this statement, (1) the number of employees of the business concern is the average over the previous fiscal year of the concern of the persons employed on a full-time, part-time or temporary basis during each of the pay periods of the fiscal year, and (2) concerns are affiliates of each other when either, directly or indirectly, one concern controls or has the power to control the other, or a third party or parties controls or has the power to control both.

I hereby declare that rights under contract or law have been conveyed to and remain with the small business concern identified above with regard to the above identified invention described in:

☒ the specification filed herewith with title as listed above.

☐ the application identified above.

☐ the patent identified above.

If the rights held by the above-identified small business concern are not exclusive, each individual, concern or organization having rights to the invention is listed on the next page and no rights to the invention are held by any person, other than the inventor, who could not qualify as an independent inventor under 37 CFR 1.9(c) or by any concern which would not qualify as a small business concern under 37 CFR 1.9(d) or a nonprofit organization under 37 CFR 1.9(e).

Each person, concern or organization to which I have assigned, granted, conveyed, or licensed or am under an obligation under contract or law to assign, grant, convey, or license any rights in the invention is listed below:

☒ no such person, concern or organization exists.
☐ each such person, concern or organization is listed below.

FULL NAME
ADDRESS
☐ Individual ☐ Small Business Concern ☐ Nonprofit Organization

FULL NAME
ADDRESS
☐ Individual ☐ Small Business Concern ☐ Nonprofit Organization

FULL NAME
ADDRESS
☐ Individual ☐ Small Business Concern ☐ Nonprofit Organization

FULL NAME
ADDRESS
☐ Individual ☐ Small Business Concern ☐ Nonprofit Organization

Separate verified statements are required from each named person, concern or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27)

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate. (37 CFR 1.28(b))

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, any patent issuing thereon, or any patent to which this verified statement is directed.

NAME OF PERSON SIGNING: Tim Williams
TITLE OF PERSON SIGNING OTHER THAN OWNER: President & Chief Executive Officer
ADDRESS OF PERSON SIGNING: CrosStor Software, Inc.
4041 Hadley Road
South Plainfield, NJ 07080

SIGNATURE: [illegible] DATE: 10/4/00

TITLE: SELF-DESCRIBING FILE SYSTEM

Inventor: Stephen A. Rago

CROSS REFERENCE TO RELATED APPLICATIONS

This application is related to, and claims the priority of, US Provisional patent application Serial Number 60/157,777 filed October 5, 1999 and entitled "Self-Describing File System" by the same inventor Stephen A. Rago.

BACKGROUND OF THE INVENTION

1. FIELD OF THE INVENTION.

The invention relates to the shared access of a computer system's file system storage by disparate, possibly unrelated, applications, such as portable file system administrative tools or sharing file systems in a storage area network (SAN).

2. DESCRIPTION OF RELATED ART.

A "file system" is an abstraction a computer operating system uses to ease the management of its user's data. Data are separated into storage units called "files" based on subject matter. Related files can be grouped together (usually also by subject matter) by listing their names in the same "directory."

Applications that need to read or write files do so through a "file system driver." The driver translates an application's request into the operations needed to read or write the storage locations that contain the data. The storage medium is usually some sort of magnetic or optical disk, but need not be limited to disks. For example, a file system driver can use RAM as the backing store for temporary file storage.

Applications usually don't know how their data are stored on disk, and don't want to know, for that matter. It is much better to isolate the knowledge of the file system format in some external place (the driver, in this case) than to embed it in each application. This makes the applications smaller, easier to write, and more portable. The benefits of portability are not to be underestimated. Many different file system formats exist, and it would be next to impossible to embed knowledge about each one in an application.

In addition to portability, centralizing the control in the file system provides a convenient way to serialize access to the on-disk data structures. If each application were to attempt to manage the file system data structures on disk, they would need to agree amongst themselves so that only one application modifies the same on-disk structure at a time. The file system driver relieves applications from having to worry about this task.

Thus, applications have evolved to ignore, for the most part, how their files are stored on disk. Nonetheless, there are still some cases where applications need to understand the on-disk file system format. Obviously, the tools used to create a file system or check a file system's consistency need to understand it's format. They are implicitly tied to the file system format, but other, more generic, applications might also need to be able to interpret the file system on-disk data structures.

For example, consider a conventional backup application that relies on the file system driver to interpret the file system format. The backup application searches the file system to copy all files to some backup medium. As each file is read, the file's access time is updated. This interferes with attempts to identify files that haven't been used for long periods of time. An administrator might wish to archive the stale files and remove them from the disk, since they are taking up disk space that might otherwise be available to store files that are accessed more frequently.

The backup application makes this difficult to do. Of course, the backup application could save the access time before reading a file, and then restore the access time after it has finished copying the file to the backup medium. However, what if someone other than the backup application reads the file while it is being backed up? The step of restoring the access time can wipe out the change to the access time that occurred when the file was read by someone else. This can lead to the file being archived prematurely.

A possible solution is to have the backup application read the disk device containing the file system and interpret the file system data structures. This avoids the updated access time, but makes the backup application specific to this file system format. A software vendor wants to write the backup application once and avoid customizing it for each different file system format.

Although others have created self-describing files, no one has attempted to create a self-describing file system. U.S. Patent No. 5,640,559 describes a way to encode file data and relationships among data in a self-describing format to allow them to be transmitted between computers more efficiently. Another example is the Hierarchical Data Format (HDF) defined by the National Center for Supercomputing Applications (NCSA). See NCSA *HDF5 Reference Manual*, Release 1.2, October 1999. It is a data format specification and a set of libraries used to create self-describing data files. It is commonly used to store scientific data.

Self-describing files have been used in a wide variety of applications including encoding data for communication between computer systems (U.S. Pat. No. 5,257,369), encoding the data in a storage dump (U.S. Pat. No. 5,761,739), a self-describing database management system (U.S. Pat. No. 5,857,195), storing the state of objects in object-oriented systems (U.S. Pat. No. 5,905,987), and encoding file objects in a distributed computing environment (U.S. Pat. No. 5,768,532).

Before it was acquired by Microsoft, Entropic, Inc. produced a library that encoded speech files in a self-describing way. Although their documentation referred to the "ESPS File System," theirs was a library that could create a set of files, and was not a general-purpose file system as previously described. For more information see Entropic Research Laboratory, Inc. *ESPS/waves+ with EnSig™ Application Notes.* Chapter entitled *"Non-ESPS Programs and the ESPS File System."* Release 5.3, 1998. http://www.ling.ed.ac.uk/help/entropic.

U.S. Patent No. 5,950,203 describes a system with improved access to data stored on a peripheral device. This applies to computer systems that can access the same storage resources on a Storage Area Network (SAN). In the method disclosed, the server is the entity that determines a file's block list. In a self-describing file system, however, the clients can determine the block list themselves.

Overall, the prior art does not appear to suggest or describe a system capable of attaining the same levels of portability and efficiency as the self-describing file system described herein.

SUMMARY OF THE INVENTION

Briefly described, the invention comprises a disk containing a file system and one or more computer systems that can access the disk. Along with the file system, the disk contains a formal description that allows applications to understand the format of the file system.

Instead of relying on a single file system driver to interpret the format of the file system stored on disk, intelligent applications can parse the structure themselves using the formal description to improve performance and add functionality. The added functionality has the advantage of being portable between different file system formats, since applications no longer need to embed knowledge of a specific format.

In a Storage Area Network, multiple computer systems can access the same set of disks. The formal description can provide the basis for accessing remote files more efficiently than existing methods, leading to better overall performance.

These and other features of the invention will be more fully understood by reference to the following drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a diagram illustrating how application programs gain access to data stored on disk through standard operating system components according to the preferred embodiment of the invention.

FIG. 2 is a diagram of several client computer systems connected to a server computer system by both a local area network and a storage are network.

FIG. 3 is a timeline illustrating the sequence of events that occur when accessing a file via a remote file system.

FIG. 4 is a timeline illustrating the sequence of events that occur when accessing a file using the proposed SNIA SAN extensions.

FIG. 5 is a timeline showing the sequence of events that occur when accessing a file stored on a self-describing file system on the SAN.

FIG. 6 is a table that compares different file system operations, showing where the responsibility lies for each operation.

FIG. 7 is the first half of an example of a partial specification of a self-describing file system.

FIG. 8 is the second half of an example of a partial specification of a self-describing file system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

During the course of this description like numbers will be used to identify like elements according to the different views which illustrate the invention.

The preferred embodiment of the invention is illustrated in Figure 1. A normal application 10 (one that does not need to interpret the file system format) reads data from and writes data to files in the operating system's file system via path 14-18. This path goes from the application into the operating system traversing the system call interface 22, the file system driver 24, and the device driver 26. The file system driver 24 understands the layout of the file system on disk 28. The disk driver 26 knows how to read and write sectors of the disk 28 and does not interpret the file system layout. With a self-describing file system, however, a more intelligent application 12 can be designed. It reads data from and writes data to files in the operating system's file system via path 16-20. This path goes from the application into the operating system traversing the system call interface 22 and the device driver 26. The description of the file system is stored on disk 28 in a well-known location, such as the last 16 KB of disk 28 containing the file system. The application 12 will read this description and use it to parse the file system data structures.

On disks controlled by the Unix operating system, files are represented by *"inodes"* (short for "index nodes"). See Bach, Maurice J., *The Design of the Unix Operating System*, Prentice-Hall, Englewood Cliffs, NJ. 1986. Each inode is identified by its *"inode number."* A formal description needs to convey how inodes are found on disk and how to extract information from the inode. Applications then need to convert the formal description into the appropriate actions to read the inodes.

Three common ways to store inodes on disk are in a table, in a segmented table (i.e., the table is split up into chunks spread out across the file system), or simply segmented (each inode is stored separately, unrelated to the location of other inodes). The inodes themselves can take on any format imaginable. For example, an inode can list the disk blocks constituting the file in a classic direct, single-indirect, double-indirect, triple-indirect layout. Other common ways to list a file's disk blocks include tree structures such as B-trees.

File systems also differ in how they associate disk blocks with files. Simple file systems use a fixed block size. For example, the System V file system has a fixed block size of either 512, 1024, or 2048 bytes. The block size is chosen when the file system is created, and all blocks in the file system are the same size. More complex file systems use variable block sizes. These are called *"extents."* Extents are contiguous ranges of disk blocks treated as one logical block. With a fixed block size, the inode need only contain the disk block addresses. With extents, the inode also needs to record the size of each extent.

A file's security information is usually stored in the file's inode, but this is not always true. Access Control Lists (ACLs) are sometimes stored separately to facilitate sharing and to provide compatibility with existing file system formats.

File system formats vary widely. This makes it difficult to develop formal descriptions. The description language needs to be flexible and general enough to support all of the different ways file systems represent and store inodes.

One alternative to a formal description is to use the algorithms actually needed to find and interpret the on-disk data structures. Presenting the algorithms to applications requires that the applications be able to execute the algorithms. This implies that the algorithm should be transmitted in some language capable of being interpreted, such as C, basic, or java

The drawbacks to using the algorithms are that the applications would need to use an interpreter to implement the algorithm, and there would probably be a performance penalty to using interpreted code instead of compiled code. The advantage, of course, is that an algorithm is the most expedient way to specify the file system format.

Although Storage Area Networks (SANs) provide high-speed access to disk storage, SANs present new problems for client access to structured on-disk data. Instead of relying on a server to read and write data on the disk, clients can directly access the disk themselves. This means that all clients must agree on the same on-disk format. This also means that the clients must use mechanisms to serialize access to the disk.

Figure 2 shows three clients 34, 36, and 38 and a server 32 connected via a local area network (LAN) 42. The file systems stored on the disk are shared by the server 32. The disk 30 is accessible to the server 32 and the clients 34, 36, and 38 via the SAN 40. Note that the SAN protocol (FCP) is different than the LAN protocol (TCP/IP).

Network file system protocols, such as NFS and CIFS, allow a server to isolate clients from the server's on-disk file system format, while still allowing the clients to access the file system through the server. This is a network-attached storage (NAS) model. The server acts as a middle-man in this model. To read or write a file, the client asks the server to perform the operation on its behalf. The drawback is reduced performance. Besides the extra delay introduced by the server (copying the data across two different communication pathways), the server becomes a bottleneck in this model when faced with concurrent requests.

Figure 3 shows the message exchange in a network file system. A client contacts the server when the network file system is mounted (step 1). The server acknowledges the request in step 2. From that point on, files can be read and written. In step 3, the client sends a read request to the server. The server responds with a message containing the results of the read request, along with any data read (step 4). In step 5, the client sends a message to the server requesting that a file be written. The data to be written to the file are contained in the write request. When the request is complete, the results are sent back to the client in step 6.

Clustered file systems provide an alternate mechanism for client access to on-disk file systems. However, clustered file systems have several drawbacks. They are complex to implement and generally don't work in heterogeneous computing environments.

The Storage Network Industry Association (SNIA) has proposed simple enhancements to conventional remote file system protocols to merge the simplicity and flexibility of the NAS model with the high performance of the SAN model. For more detail see the following publications by CrosStor, Inc. a. CIFS Extensions for SANs, and b. Adapting NAS Protocols to SANs, both White Papers published May 10, 1999. In the proposals, clients request a server to perform all file operations except read and write. To read or write a file, a client will only contact the server to identify the physical blocks on disk to read or write. With the list of physical blocks, a client can then access the disks directly. This proposal has the benefit of removing the need for the clients to understand the file system format. The drawback is that the server can still be a bottleneck when translating an (offset, length) pair in a file to a list of physical block numbers.

Figure 4 illustrates the message exchange in the SNIA proposal. Mounts (steps 1 and 2) occur in the same way as in the standard network file system case. The differences show up in the reads and writes. Read requests are replaced by requests to translate the range of the file to be read into a list of physical block numbers (step 3). The server parses the on-disk data structures for the file and returns to the client a list of physical blocks corresponding to the portion of the file to be read (step 4). In step 5, the client uses the list of block numbers to read the data across the SAN, accessing the disk directly. A similar process occurs for writes (steps 6, 7, and 8). Note that the requests and responses exchanged between the client and server go across the LAN. The SAN is only used to access the disk. Newer advances in technology allow both SAN and LAN traffic to use the same physical wire (communication pathway), but this does not affect the operation of the invention.

The SNIA proposal is similar in spirit to the method described in U.S. Patent No. 5,950,203, although the SNIA proposal is much simpler. The patent disclosure differs in that the disk is used to store the block list for the file being read or written in a different, temporary file. Instead of receiving the block list in a network message, the client reads the list from the temporary file stored on disk.

A self-describing file system can be accessed by clients across a SAN, but in a portable and extensible manner. As in the SNIA proposal, each file system is shared using a NAS protocol. Unlike the SNIA proposal, however, the server does not supply clients with a list of physical block numbers. Instead, when the client attaches to the network file system, the client reads a formal description of the file system from the disk containing the file system, with enough information to allow the client to determine:

1. The physical block and offset containing a file's inode given it's inode number. This is basically the "iread" algorithm typically found in Unix file system drivers.

2. The block list given an offset into the file and a length. This is basically the "bmap" algorithm typically found in Unix file system drivers.

With the formal description, an arbitrary client can determine the blocks to access when reading and writing a file. The server still needs to perform block allocation and file serialization, though. The client need not change when the file system format changes, because the formal description that is stored on disk will change as the file system format changes.

Figure 5 shows the message exchange that occurs between a client and server using self-describing file systems. After the mount succeeds (steps 1 and 2), the client reads a formal description of the file system from the disk (step 3). The client saves this description for use in the future. When a read request (step 4) or a write request (step 5) occurs, the client uses the formal description to interpret the file system format and reads or writes the data across the SAN directly.

Figure 6 illustrates the differences between the alternate configurations. Each cell of the table identifies who is responsible for the various aspects of file system operation. At one extreme, a network file system places all responsibility with the server. At the other extreme, a clustered file system relies on the clients to do everything for themselves (there really isn't a server per se). Between these two extremes are the current SNIA proposal and the self-describing file system invention described here. Self-describing file systems allow clients to perform the block mapping themselves without requiring the clients to build in knowledge about the file system format.

The System V file system (commonly known as S5) uses a 1KB block size. More detail about the file system format can be found in Bach, Maurice J., *The Design of the Unix Operating System*, Prentice-Hall, Englewood Cliffs, NJ. 1986 previously discussed. The first block contains optional boot code followed by the super block. The super block contains global information about the file system. After the super block is a contiguous set of blocks containing the array of inodes. The remaining set of blocks are either associated with files or linked on a list of available blocks.

Figures 7 and 8 show the C code, specified using XML, needed to find a particular inode in the file system and translate (offset, length) pairs into a list of physical block numbers. This is a portion of a formal description, specified in C, that would be necessary to allow an application to parse the file system format. (The algorithms assume that the applications provide a routine called PREAD to read a disk block. Error cases are ignored for simplicity in the example.)

Simple backup programs read files by going through the file system driver, as discussed in the background. This can cause problems by updating the access time unintentionally. More intelligent backup programs read the disk directly, thereby bypassing the file system driver. These backup programs would have to contain information about which file system formats they support, but with self-describing file systems, these programs can be made portable and can support many different formats just by supporting the ability to interpret the formal description.

Performing "invisible" reads and writes is not the only motivation for reading the disk directly. By bypassing the file system driver, a backup program can improve its performance. Even more performance increases can be realized during incremental backups. Instead of searching the file system, the on-disk inode table can be scanned for candidate files. Scanning a table is considerably faster than searching directories and checking modification times by calling stat (2) on each file.

Another benefit of self-describing file systems is that most file system utility commands can be made independent of the file system type by using the file system's formal description. Unix commands such as df, fstyp, labelit, and ncheck could be written once instead of developing a different version for each supported file system format.

Self-describing file systems allow applications to be developed that can understand multiple file system formats in an extensible and flexible manner. When a file system format evolves, applications do not need to be modified. Instead, only the formal description of the file system needs to be changed. Additionally, a single application can work with different file system formats, because it can adapt dynamically based on the file system's formal description.

While the invention has been described with reference to the preferred embodiment thereof it will be appreciated by those of ordinary skill in the art that modifications can be made to the parts that comprise the invention without departing from the spirit and scope thereof.

WE CLAIM:

Claim 1. A method for use in a self-describing file system including at least one server and one disc storage device for access by at least one client including, said method comprising the steps of:

a. attaching said client to said file system; and,

b. reading a formal description of the file system by said client from said disc storage device,

wherein said client can substantially directly read and write files or blocks of data to and from said disc storage device without requiring further knowledge of said file system.

Claim 2. The method of claim 1 wherein step b. further comprises the step of:

c. reading enough information to determine the physical block and offset containing a given file's inode given its inode number.

Claim 3. The method of claim 2 wherein step b. further comprises the step of:

d. reading enough information to determine the block list of a given file given an offset into the file and a length.

Claim 4. The method of claim 3 wherein step c. comprises reading the iread algorithm found in a Unix file system.

Claim 5. The method of claim 4 wherein step d. comprises the step of reading the bmap of a Unix file system.

Claim 6. The method of claim 5 wherein step a. comprises the steps of:

e. sending a mount request; and,

f. receiving a mount response.

Claim 7. The method of claim 6 wherein said formal description of the file system read in step b. is saved for future use when a read request or a write request is made by said client.

Claim 8. The method of claim 7 wherein said disc storage device is located in a Storage Area Network (SAN).

Claim 9. The method of claim 7 wherein said client is located on said server.

Claim 10. The method of claim 1 wherein said reading step b. comprises reading for backup and restore purposes.

Claim 11. A self- describing file system comprising:

disc storage means for storing files and data;

file server means attachable to said disc storage means for accessing said disc storage means;

client means connected to said file server means; and,

formal description obtaining means within said client means for obtaining a formal description of the file system from the disc storage means containing the file system,

wherein said client can substantially directly read and write files or blocks of data to and from said disc storage means without requiring further knowledge of said file system.

Claim 12. The system of claim 11 wherein said formal description obtaining means includes means for reading enough information to determine the physical block and offset containing a given file's inode given its inode number.

Claim 13. The system of claim 12 wherein said formal description obtaining means further includes means for reading enough information to determine the block list of a given file given an offset into the file and a length.

Claim 14. The system of claim 13 wherein said disc storage means is located in a Storage Area Network (SAN).

Claim 15. The method of claim 13 wherein said client is located on said server.

ABSTRACT OF THE DISCLOSURE

The invention provides a way for computer applications to parse the operating system's file system format without embedding direct knowledge of the format in the applications themselves. By making a file system self-describing, applications running locally on the same computer, or remotely on another computer, can interpret file system data structures if they can access the disk containing the file system. Storage Area Networks (SANs) present a paradigm where multiple computer systems can see the same set of disk resources. This, combined with the invention of self-describing file systems, makes it possible to build applications that are more intelligent and perform better than their counterparts that either embed knowledge of a file system or rely on a file system driver to interpret the structure on behalf of the applications.

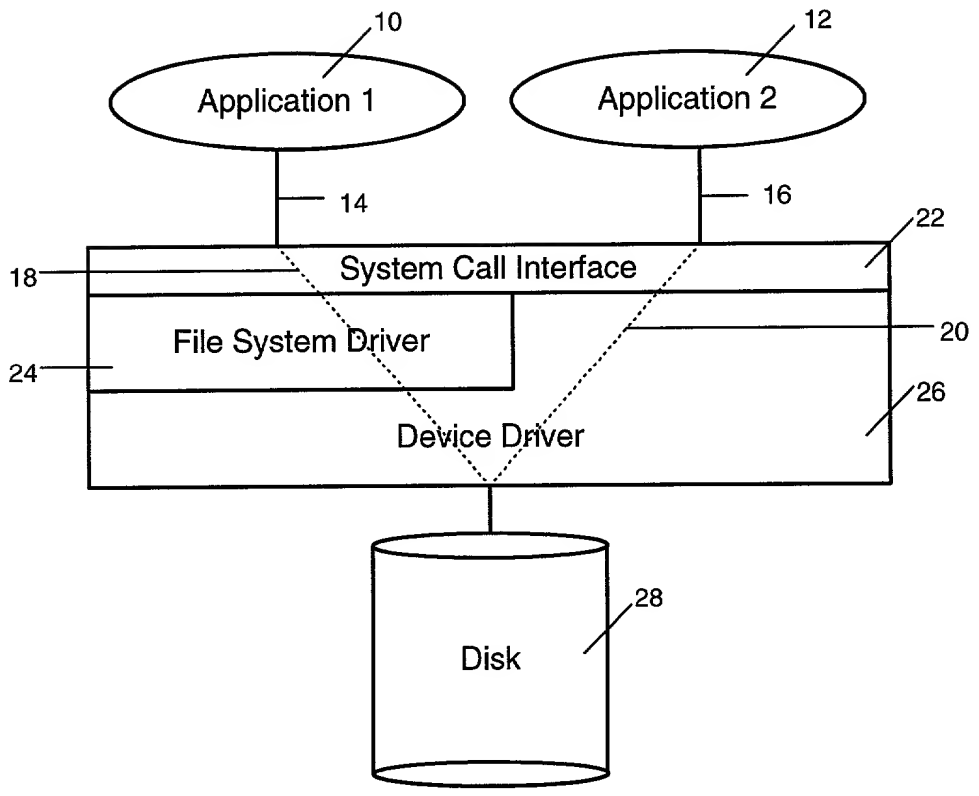
10
12
Application 1
Application 2
14
16
22
18
System Call Interface
File System Driver
20
24
26
Device Driver
28
Disk

Figure 1.

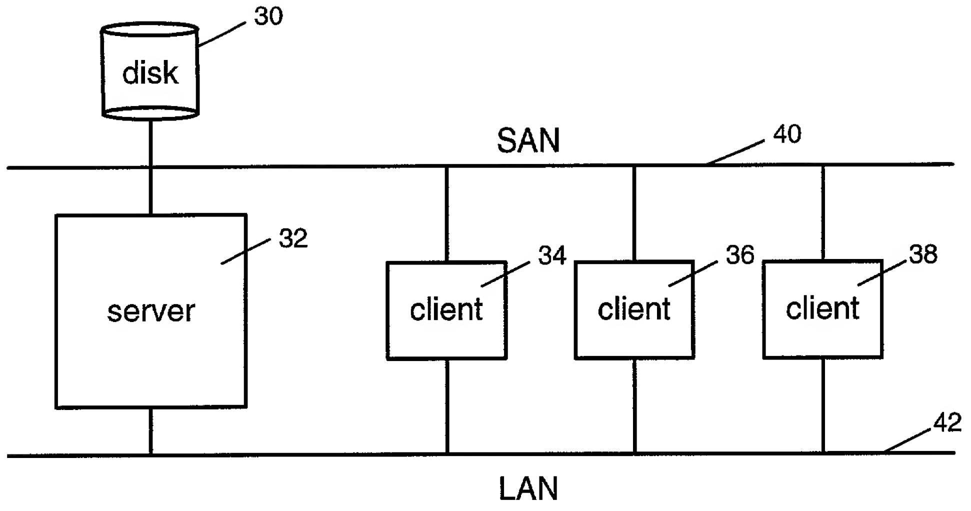

Figure 2.

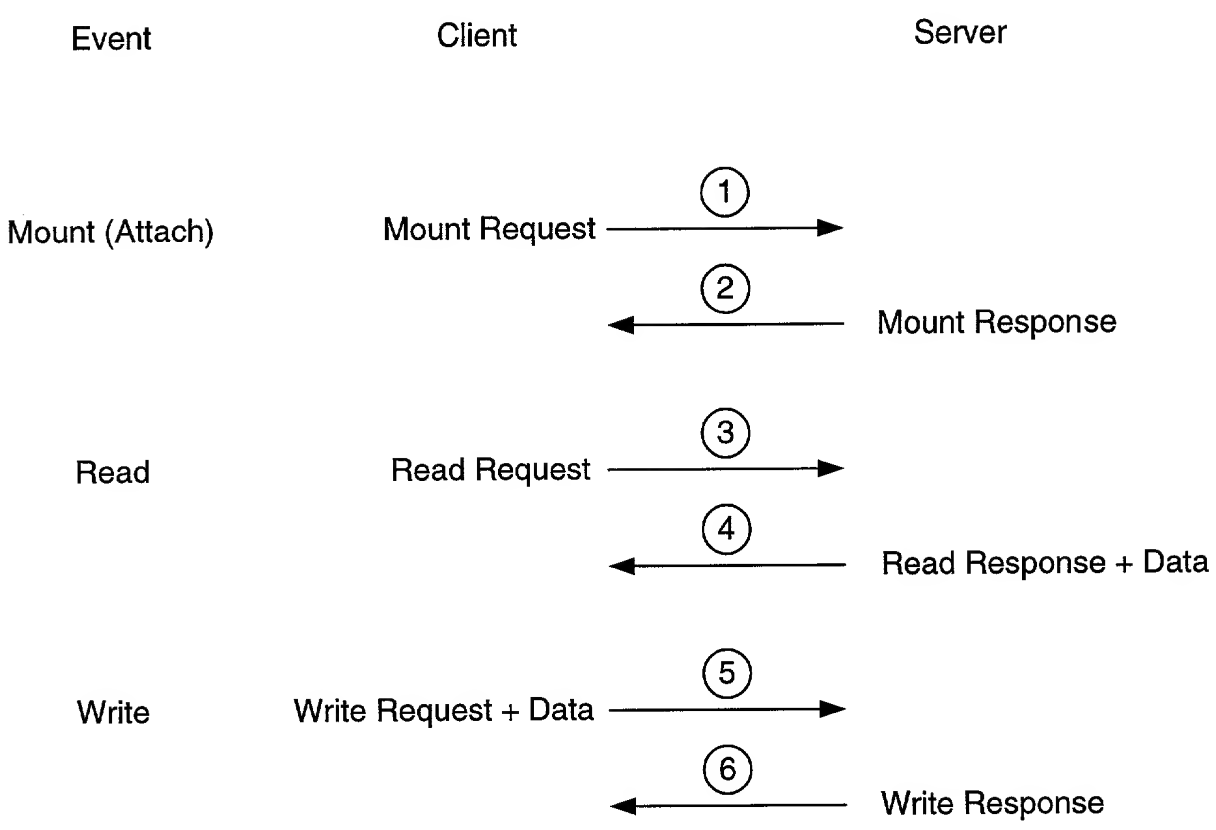
Event
Client
Server
Mount (Attach)
Mount Request
1
2
Mount Response
Read
Read Request
3
4
Read Response + Data
Write
Write Request + Data
5
6
Write Response

Figure 3.

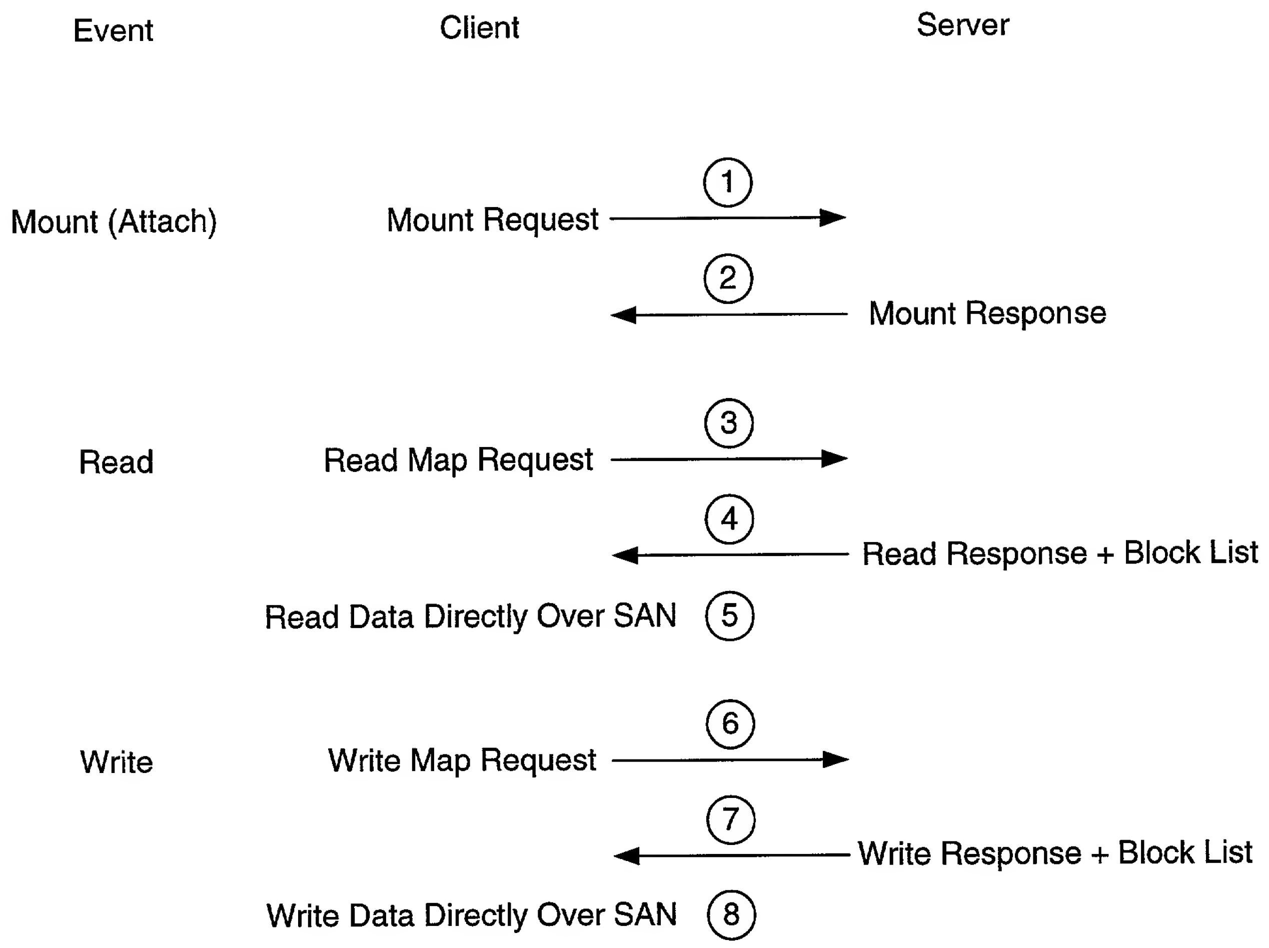
Event
Client
Server
Mount (Attach)
Mount Request
1
2
Mount Response
Read
Read Map Request
3
4
Read Response + Block List
Read Data Directly Over SAN
5
Write
Write Map Request
6
7
Write Response + Block List
Write Data Directly Over SAN
8

Figure 4.

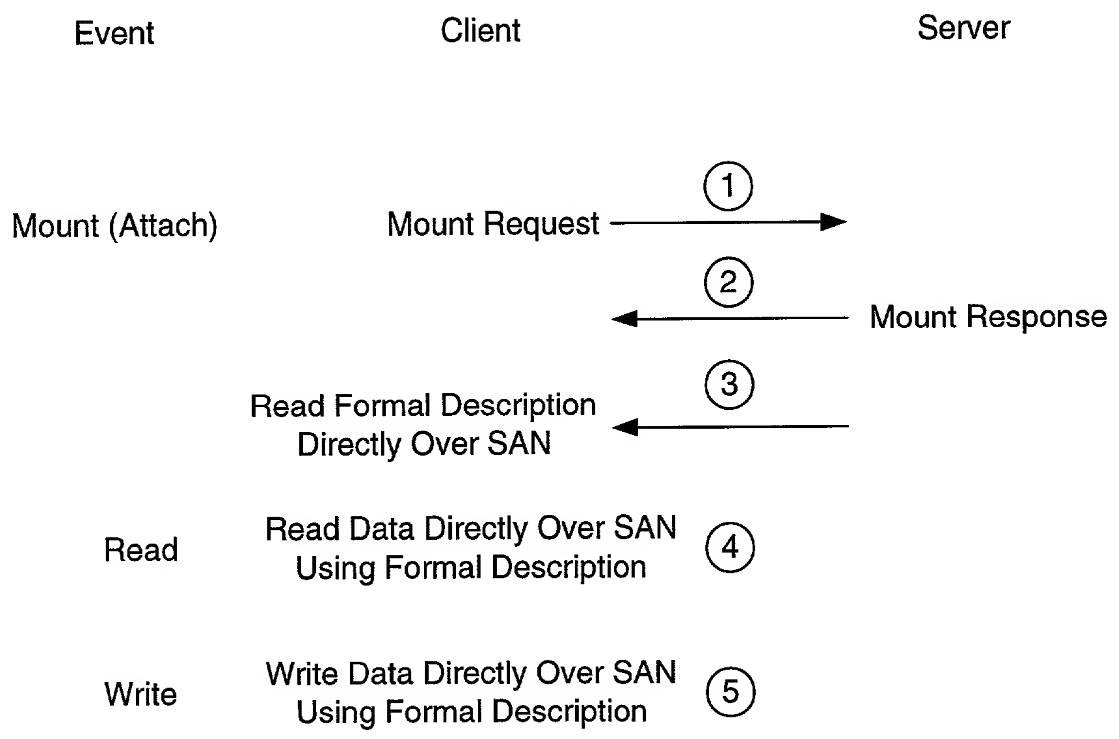
Event
Client
Server
Mount (Attach)
Mount Request
1
2
Mount Response
3
Read Formal Description Directly Over SAN
Read
Read Data Directly Over SAN Using Formal Description
4
Write
Write Data Directly Over SAN Using Formal Description
5

Figure 5.

Feature	Network File System	SNIA Proposal	Self-Describing File System	Clustered File System
Serialization	server	server	server	clients
Disk Block Mapping	server	server	clients	clients
Block Allocation	server	server	server	clients
Disk Access	server	clients	clients	clients

Figure 6.

```
<FS Parameters>
      <CONST NAME="BSIZE"> 1024 </CONST>
      <CONST NAME="INOSZ"> 64 </CONST>
      <CONST NAME="INOPB">  BSIZE / INOSZ </CONST>
      <CONST NAME="ISIZE"> {derived from super block} </CONST>
      <CONST NAME="STARTI"> 2 </CONST>
      <CONST NAME="PSTART"> {physical start of file system} </CONST>
      <CONST NAME="NADDR"> 13 </CONST>
      <CONST NAME="NDADDR"> 10 </CONST>
      <CONST NAME="NIADDR"> 3 </CONST>
      <CONST NAME="IADDRSZ"> 3 </CONST>
      <CONST NAME="IADDROFF"> 12 </CONST>
      <CONST NAME="ISIZEOFF"> 8 </CONST>
      <CONST NAME="BYTEORDER"> 0 </CONST>
      <CONST NAME="NSHIFT"> 8 </CONST>
      <CONST NAME="NINDIR"> BSIZE / 4 </CONST>
      <CONST NAME="NBPSCTR"> 512 </CONST>
</FS Parameters>

<MACRO NAME="LTOPBLK" PARAMS="BN"> BN * (BSIZE / NBPSCTR) </MACRO>

<FUNC NAME="iread">
      <BODY>
            int32
            iread(int16 ino, char *buf)
            {
                  int32 bn;
                  int32 boff;

                  bn = (ino + (2 * INOPB - 1)) / INOPB;
                  boff = (ino + (2 * INOPB - 1)) & (INOPB - 1);
                  PREAD(LTOPBLK(bn), buf, BSIZE);
                  return boff;
            }
      </BODY>
</FUNC>

<FUNC NAME="bmap">
      <BODY>
            int32
            bmap(char *ibuf, int32 off, int32 len, int32 *dbuf)
            {
                  int32 sh;
                  int32 i;
                  int32 j;
                  int32 bn;
                  int32 blim;
                  int32 nblk;
                  int32 *bnp;
                  int32 daddr[NADDR];
                  char ib[BSIZE];
                  char *cp;
                  int32 naddr = 0;

                  nblk = len + (BSIZE - 1) / BSIZE;
                  if (nblk == 0)
                        return 0;
```

Figure 7.

```
                    /*
                     * build an address array, converting from 3-byte
                     * addresses to 4-byte addresses.
                     */
                    cp = ibuf + IADDROFF;
                    for (i = 0, j = 0; i < IADDRSZ; i += IADDRSZ, j++) {
                            if (BYTEORDER == 0)
                                    daddr[j] = cp[i]<<16|cp[i+1]<<8|cp[i+2];
                            else
                                    daddr[j] = cp[i+2]<<16|cp[i+1]<<8|cp[i];
                    }

                    /*
                     * Fill the dbuf array with the list of block numbers.
                     */
                    while (len > 0) {
                            bn = off / BSIZE;
                            len -= BSIZE;
                            off += BSIZE;
                            if (bn < NADDR - NIADDR) {
                                    dbuf[naddr++] = daddr[bn];
                                    continue;
                            }
                            bn -= NDADDR;
                            sh = 0;
                            blim = 1;
                            for (j = NIADDR; j > 0; j--) {
                                    sh += NSHIFT;
                                    blim <<= NSHIFT;
                                    if (bn < blim)
                                            break;
                            }
                            if (j == 0)
                                    return naddr;
                            ibn = daddr[NADDR-j];
                            if (inb == 0) {
                                    dbuf[naddr++] = 0;
                                    continue;
                            }
                            for (; j <= 3; j++) {
                                    sh -= NSHIFT;
                                    PREAD(LTOPBLK(ibn), ib, BSIZE);
                                    bnp = (int32 *)ib;
                                    i = (bn >> sh) & (NINDIR - 1);
                                    if (bnp[i] == 0)
                                            break;
                                    ibn = bnp[i];
                            }
                            dbuf[naddr++] = bnp[i];
                    }
                    return naddr;
            }
        </BODY>
</FUNC>
```

Figure 8.

Docket No.
3728-109US

Declaration and Power of Attorney For Patent Application

English Language Declaration

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

SELF-DESCRIBING FILE SYSTEM

the specification of which

(check one)

☒ is attached hereto.

☐ was filed on ________________ as United States Application No. or PCT International Application Number ______________________________ and was amended on ______________________________
(if applicable)

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose to the United States Patent and Trademark Office all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119(a)-(d) or Section 365(b) of any foreign application(s) for patent or inventor's certificate, or Section 365(a) of any PCT International application which designated at least one country other than the United States, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate or PCT International application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application(s)			Priority Not Claimed
(Number)	(Country)	(Day/Month/Year Filed)	☐
(Number)	(Country)	(Day/Month/Year Filed)	☐
(Number)	(Country)	(Day/Month/Year Filed)	☐

I hereby claim the benefit under 35 U.S.C. Section 119(e) of any United States provisional application(s) listed below:

60/157,777	10/05/1999
(Application Serial No.)	(Filing Date)
(Application Serial No.)	(Filing Date)
(Application Serial No.)	(Filing Date)

I hereby claim the benefit under 35 U. S. C. Section 120 of any United States application(s), or Section 365(c) of any PCT International application designating the United States, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of 35 U.S.C. Section 112, I acknowledge the duty to disclose to the United States Patent and Trademark Office all information known to me to be material to patentability as defined in Title 37, C. F. R., Section 1.56 which became available between the filing date of the prior application and the national or PCT International filing date of this application:

(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. *(list name and registration number)*

Richard C. Woodbridge	**26,423**
Stuart H. Nissim	**33,351**
Kane Koo	**44,849**

Send Correspondence to: **Richard C. Woodbridge, Esq.**
Woodbridge & Associates, P.C.
P.O. Box 592
Princeton, NJ 08542-0592

Direct Telephone Calls to: *(name and telephone number)*
Richard C. Woodbridge, Esq. **609-924-3773**

Full name of sole or first inventor **STEPHEN A. RAGO**	
Sole or first inventor's signature Stephen A. Rago	Date 10-3-00
Residence **Berkely Heights, NJ 07922**	
Citizenship **USA**	
Post Office Address **198 Spring Ridge Drive, Berkeley Heights, New Jersey 07922**	

Full name of second inventor, if any	
Second inventor's signature	Date
Residence	
Citizenship	
Post Office Address	

United States Patent & Trademark Office

Office of Initial Patent Examination -- Scanning Division

Application deficiencies were found during scanning:

☐ Page(s)_________ of ____________________________ were not present for scanning. (Document title)

☐ Page(s)_________ of ____________________________ were not present for scanning. (Document title)

☐ Scanned copy is best available. Verified Statement